Default Access in Subclasses - Complete Notes

Key Question: Can subclasses access default members and methods?

Answer: YES, but ONLY if the subclass is in the SAME PACKAGE as the superclass.

Understanding Default Access

Default access (no access modifier keyword) provides **package-level visibility**, not inheritance-based visibility.

Rule:

- **Same Package**: Default members are inherited and accessible in subclasses ✅
- **Different Package**: Default members are NOT accessible in subclasses ❌

Access Modifier Inheritance Table

Access Modifier	Same Class	Same Package	Subclass (Same Package)	Subclass (Different Package)	Different Package
private	✅	❌	❌	❌	❌
default	✅	✅	✅	❌	❌
protected	✅	✅	✅	✅	❌
public	✅	✅	✅	✅	✅

Example 1: Same Package (Default members ARE accessible)

Parent Class:

```java
```

```java
// File: mypackage/Vehicle.java
package mypackage;

public class Vehicle {
    public String brand;         // public - accessible everywhere
    protected int maxSpeed;      // protected - accessible in subclasses
    String fuelType;             // default - accessible in same package
    private String engineNumber; // private - not accessible in subclasses

    public void start() {
        System.out.println("Vehicle starting");
    }

    protected void accelerate() {
        System.out.println("Vehicle accelerating");
    }

    void refuel() {              // default method
        System.out.println("Vehicle refueling with " + fuelType);
    }

    private void performMaintenance() {
        System.out.println("Performing maintenance");
    }
}
```

Subclass in Same Package:

```java
```

```java
// File: mypackage/Car.java
package mypackage;              // SAME PACKAGE

public class Car extends Vehicle {
    private int doors;

    public Car(String brand, int maxSpeed, String fuelType, int doors) {
        this.brand = brand;      // ✅ public - accessible
        this.maxSpeed = maxSpeed; // ✅ protected - accessible in subclass
        this.fuelType = fuelType; // ✅ default - accessible (same package)
        this.doors = doors;
        // this.engineNumber = "123"; // ❌ private - not accessible
    }

    public void carActions() {
        start();                // ✅ public method - accessible
        accelerate();           // ✅ protected method - accessible
        refuel();               // ✅ default method - accessible (same package)
        // performMaintenance(); // ❌ private method - not accessible

        System.out.println("Car has " + doors + " doors");
        System.out.println("Fuel type: " + fuelType); // ✅ Can access default variable
    }

    // Can override default method since we're in same package
    @Override
    void refuel() {
        System.out.println("Car refueling with " + fuelType + " at gas station");
    }

    // New method specific to Car
    void openTrunk() {
        System.out.println("Car trunk opened");
    }
}
```

Example 2: Different Package (Default members are NOT accessible)

Subclass in Different Package:

java

```java
// File: anotherpackage/Motorcycle.java
package anotherpackage;          // DIFFERENT PACKAGE
import mypackage.Vehicle;

public class Motorcycle extends Vehicle {
    private boolean hasSidecar;

    public Motorcycle(String brand, int maxSpeed, boolean hasSidecar) {
        this.brand = brand;       // ✅ public - accessible
        this.maxSpeed = maxSpeed; // ✅ protected - accessible in subclass
        // this.fuelType = "Petrol"; // ❌ default - not accessible (different package)
        this.hasSidecar = hasSidecar;
    }

    public void motorcycleActions() {
        start();                  // ✅ public method - accessible
        accelerate();             // ✅ protected method - accessible
        // refuel();              // ❌ default method - not accessible (different package)

        System.out.println("Motorcycle has sidecar: " + hasSidecar);
        // System.out.println("Fuel: " + fuelType); // ❌ Cannot access default variable
    }

    // This creates a NEW method, NOT an override of parent's default method
    void refuel() {
        System.out.println("Motorcycle refueling");
        // This doesn't override parent's refuel() because parent's refuel() is not visible
    }

    void wheelie() {
        System.out.println("Motorcycle doing a wheelie!");
    }
}
```

Complete Working Example

java

```java
// File: com/example/shapes/Shape.java
package com.example.shapes;

public class Shape {
    public String color;         // public access
    protected double area;       // protected access
    String name;                 // default access
    private int id;              // private access

    public Shape(String color, String name) {
        this.color = color;
        this.name = name;
        this.id = generateId();
    }

    public void display() {
        System.out.println("Shape: " + name + ", Color: " + color);
    }

    protected void calculateArea() {
        System.out.println("Calculating area for " + name);
    }

    void showDetails() {         // default method
        System.out.println("Name: " + name + ", Area: " + area + ", ID: " + id);
    }

    private int generateId() {
        return (int)(Math.random() * 1000);
    }

    private void logInfo() {
        System.out.println("Logging info for shape " + id);
    }
}

// File: com/example/shapes/Circle.java (SAME PACKAGE)
package com.example.shapes;      // Same package

public class Circle extends Shape {
    private double radius;

    public Circle(String color, double radius) {
        super(color, "Circle");   // Call parent constructor
        this.radius = radius;
        this.area = Math.PI * radius * radius; // ✅ Can access protected variable
```

```
        // Can access default variable from same package
        System.out.println("Creating " + name + " with radius " + radius); // ✅
    }

    @Override
    protected void calculateArea() {
        this.area = Math.PI * radius * radius;
        System.out.println("Circle area calculated: " + area);
    }

    // Can override default method since we're in same package
    @Override
    void showDetails() {
        System.out.println("Circle - Name: " + name + ", Radius: " + radius + ", Area: " + area);
    }

    public void circleSpecificMethod() {
        display();          // ✅ public method
        calculateArea();    // ✅ protected method
        showDetails();      // ✅ default method (same package)
        // logInfo();       // ❌ private method - not accessible
    }
}

// File: com/example/geometry/Rectangle.java (DIFFERENT PACKAGE)
package com.example.geometry;    // Different package
import com.example.shapes.Shape;

public class Rectangle extends Shape {
    private double length;
    private double width;

    public Rectangle(String color, double length, double width) {
        super(color, "Rectangle");
        this.length = length;
        this.width = width;
        this.area = length * width;  // ✅ Can access protected variable

        // Cannot access default variable from different package
        // System.out.println("Creating " + name); // ❌ Compilation error
    }

    @Override
    protected void calculateArea() {
        this.area = length * width;
        System.out.println("Rectangle area calculated: " + area);
```

```
    }

    // This creates a NEW method, not an override
    void showDetails() {
        System.out.println("Rectangle - Length: " + length + ", Width: " + width + ", Area: " + area);
        // Cannot access 'name' variable here as it's default access from different package
    }

    public void rectangleActions() {
        display();              // ✅ public method
        calculateArea();        // ✅ protected method
        // showDetails();       // This calls the NEW showDetails() method defined in this class
        // Parent's showDetails() is not accessible from different package
    }
}
```

Testing the Examples:

java

```
// File: TestDefaultAccess.java
package com.example.shapes;     // Same package as Shape

public class TestDefaultAccess {
    public static void main(String[] args) {
        System.out.println("=== CIRCLE (SAME PACKAGE) ===");
        Circle circle = new Circle("Red", 5.0);
        circle.circleSpecificMethod();
        circle.showDetails();   // Calls overridden method

        System.out.println("\n=== RECTANGLE (DIFFERENT PACKAGE) ===");
        com.example.geometry.Rectangle rectangle =
            new com.example.geometry.Rectangle("Blue", 4.0, 6.0);
        rectangle.rectangleActions();
        rectangle.showDetails(); // Calls Rectangle's own method, not parent's
    }
}
```

Key Points to Remember:

✅ What Subclasses CAN Access:

1. **Same Package**:
   - Public members
   - Protected members

- **Default members** ✅
- Can override default methods ✅

2. **Different Package**:
   - Public members
   - Protected members
   - Cannot access default members ❌

❌ What Subclasses CANNOT Access:

- Private members (regardless of package)
- Default members from different packages

Important Rules:

1. **Package Boundaries Matter**: Default access is about packages, not inheritance
2. **Method Overriding**: Can only override default methods if in same package
3. **Variable Access**: Default variables only accessible from same package subclasses
4. **Constructor Access**: Default constructors follow same rules
5. **Different Package = New Method**: Defining a method with same name in different package subclass creates NEW method, not override

Common Mistakes:

1. **Assuming inheritance overrides package rules** - It doesn't for default access
2. **Forgetting package declarations** - Always check package statements
3. **Thinking default = protected** - They are different!
4. **Confusing method override with new method** - Same name in different package = new method

Best Practices:

1. **Use protected instead of default** if you want subclass access across packages
2. **Keep related classes in same package** if they need default access
3. **Be explicit with access modifiers** - don't rely on default
4. **Document package-private members** clearly
5. **Consider package organization** when designing inheritance hierarchies

Summary:

Default access in subclasses depends entirely on package location:

- **Same package subclass**: ✅ Full access to default members

- **Different package subclass**: ❌ No access to default members

This is a crucial concept that combines inheritance with Java's package-based access control system!